Trabajo Práctico 1:
Conjunto de instrucciones MIPS

Fabrizio Cozza, *Padrón Nro. 97.402*
fabrizio.cozza@gmail.com

Kevin Cajachuán, *Padrón Nro. 98.725*
kevincajachuan@hotmail.com

Luciano Giannotti, *Padrón Nro. 97.215*
luciano_giannotti@hotmail.com.ar

1er. Cuatrimestre de 2018
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

1. Objetivos

Este Trabajo Práctico tiene el fin de ayudarnos a familiarizarnos con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto en la siguiente sección.

2. Programa

El software de este trabajo está escrito en su mayoría en lenguaje C y permite dibujar **Julia Sets** o **Conjuntos de Julia** segun los parámetros que le pasamos por línea de comando. Estos parámetros son la región del plano complejo: delimitada por un centro, un ancho y un alto; una semilla que afectará el calculo para cada pixel; la resolución y la salida ya sea por pantalla o por archivo.
La función en la cuál se encuentra la lógica de cómputo del fractal está escrita en MIPS con el fin de tener soporte nativo para NetBSD.
El formato a usar es PGM o *portable gray format*, que resulta útil para describir imágenes digitales en escala de grises.

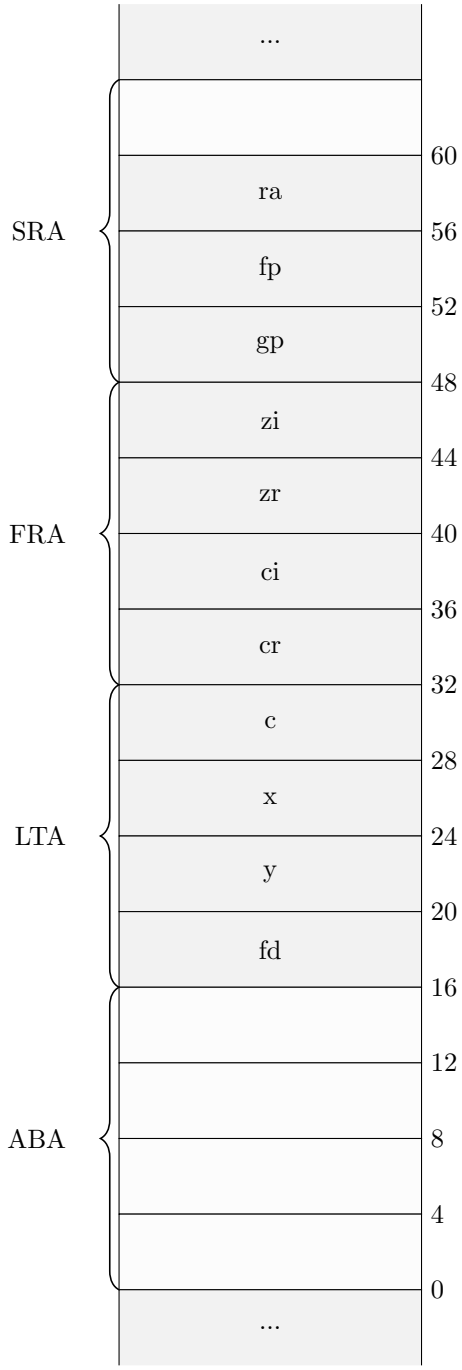
3. Implementación

Una vez recibidos los parámetros, para dibujar el Julia Set el programa convierte cada píxel de la ventana a un punto en el plano complejo. A ese punto se lo eleva al cuadrado y le suma la semilla mencionada en la sección anterior. Esto se repite hasta que el valor absoluto del resultado sea menor a 2, en cuyo caso se toma la cantidad de iteraciones y se imprime en el archivo PGM, representando el nivel de blanco de ese píxel.

3.1. Funciones implementadas

3.1.1. mips32_plot

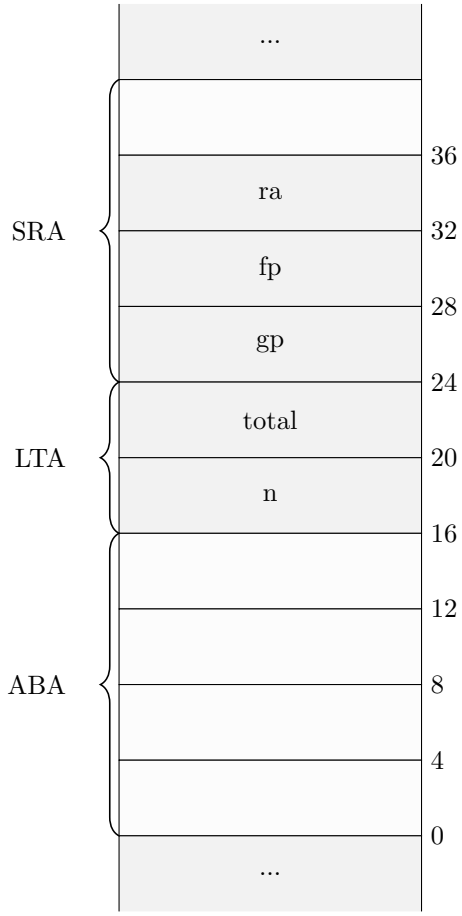
Esta función es la que se encarga de hacer los cálculos, para luego poder ir imprimiendo el valor de cada píxel en un archivo.
El único parámetro que recibe es un struct definido como *param_t* en el que se encuentran todos los datos necesarios para que la función realice su tarea, los cuales se obtienen de los parámetros pasados por el usuario por línea de comandos.
Esta función no devuelve nada ya que solo se dedica a hacer cálculos e imprimir.
El stack frame de esta función se muestra a continuación, en el cuál en cada dirección de memoria se indica qué variable está guardada.



3.1.2. my_fprintf

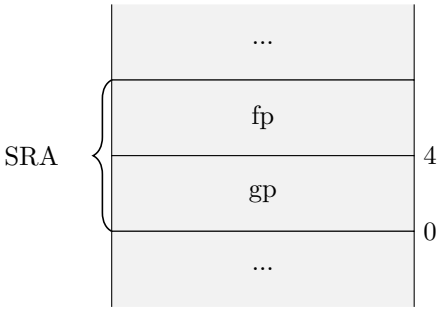
Esta función imprime en un archivo llamando a la syscall *write*, por lo que los parámetros que recibe son los mismo que recibe esta syscall: file descriptor del archivo, lo que se quiere escribir, y cuánto se quiere escribir. Finalmente devuelve la cantidad de bytes que se escribió la igual que la syscall.

El stack frame de esta función se muestra a continuación.



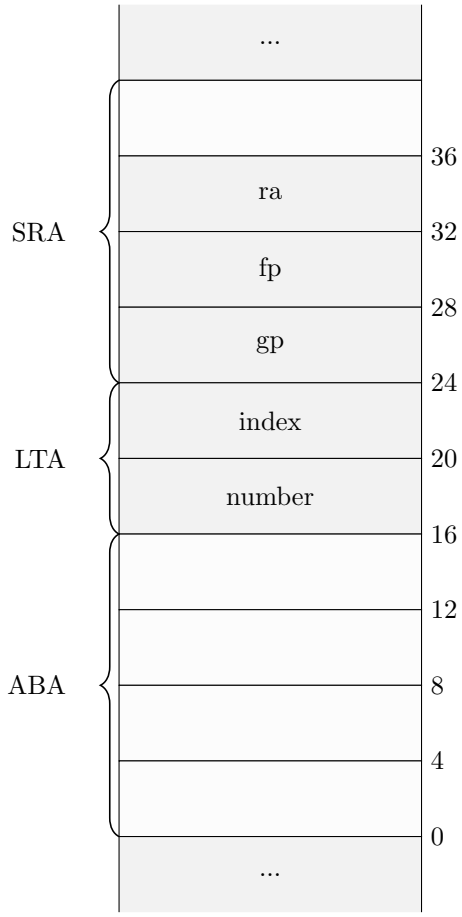
3.1.3. my_strlen

Esta función calcula la longitud de una cadena, recibiendo como parámetro la cadena y devolviendo la longitud.
El stack frame de la función se muestra a continuación. Al ser una función *leaf*, no fue necesario tener un ABA ni salvar el registro *ra*.



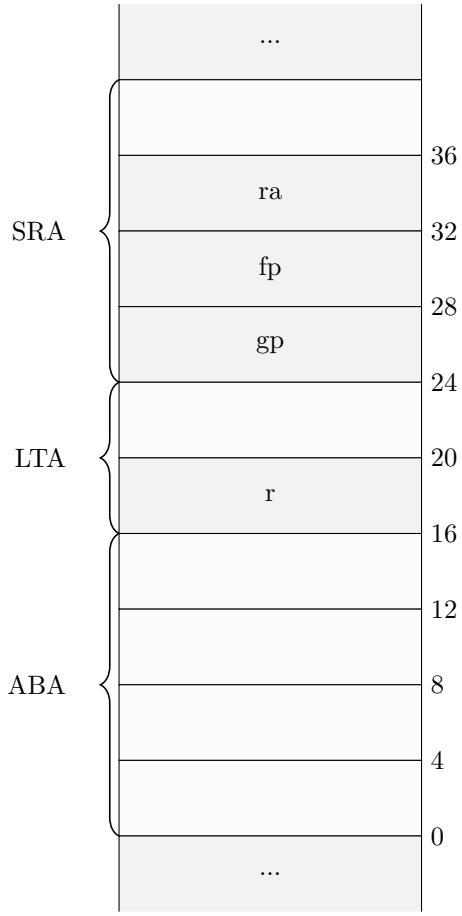
3.1.4. int_to_str

Esta función convierte un número a una cadena, llamando para realizar esto a dos funciones: *dig_to_char* y *put_end*. Lo único que recibe la función es el número que se quiere convertir y no devuelve nada.
El stack frame se muestra a continuación.



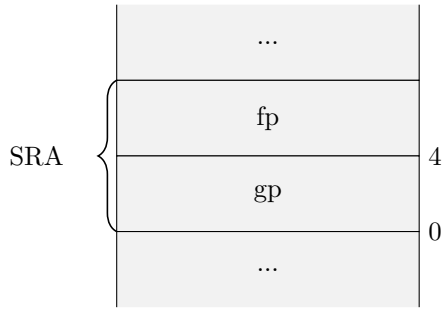
3.1.5. dig_to_char

Esta función recibe un número que se quiere convertir a cadena, un array para ir guardando los caracteres de cada dígito del número y el índice actual en el que hay que ir guardando cada caracter. Esta función se llama recursivamente para ir obteniendo los dígitos del número e ir guardandolos en el array. Como esto es lo único que realiza, la función no devuelve nada.
A continuación se muestra el stack frame de esta función.



3.1.6. put_end

Esta función lo único que hace es poner el caracter \0 al final del array en que se guardaron los dígitos del número, para poder imprimirlo más adelante. Por esta razón esta función recibe como parámetros el array y el índice en el que se tiene que guardar el caracter \0 y no devuelve ningún valor.
El stack frame se muestra a continuación. Por la misma razón que en la función *my_strlen*, no es necesario tener un ABA ni salvar *ra*.



4. Código S

En esta sección se muestra el código

```
#include <sys/syscall.h>
#include <mips/regdef.h>

          .text
          .abicalls
          .align 2
          .globl mips32_plot
          .ent mips32_plot
mips32_plot:
          .frame $fp, 64, ra
          .set noreorder
          .cpload t9
          .set reorder
          subu sp, sp, 64
          .cprestore 48
          sw $fp, 52(sp)
          sw ra, 56(sp)
          move $fp, sp
          sw a0, 64($fp)                          # Fin de creacion de
              SF
          lw t0, 44(a0)                           # Obtengo dir de fp
              en t0
          lh t0, 14(t0)                           # Obtengo fd
          sw t0, 16($fp)                          # Salvo fd
          la a0, head_pgm
          jal my_strlen
          move a2, v0                             # Cargo len(head_pgm)
          lw a0, 16($fp)                          # Cargo fd
          la a1, head_pgm                         # Cargo head_pgm
          jal my_fprintf                          # Imprimo "P2"
          lw t0, 64($fp)
          lw a0, 32(t0)                           # Cargo x_res
          jal int_to_str                          # x_res -> str
          la a0, number
          jal my_strlen
          move a2, v0                             # Cargo len(number)
          lw a0, 16($fp)                          # Cargo fd
          la a1, number                           # Cargo number
          jal my_fprintf                          # Imprimo "x_res"
          lw a0, 16($fp)                          # Cargo fd
          la a1, new_line                         # Cargo "\n"
          li a2, 1                                # Cargo len("\n")
          jal my_fprintf                          # Imprimo "\n"
          lw t0, 64($fp)
          lw a0, 36(t0)                           # Cargo y_res
          jal int_to_str                          # y_res -> str
          la a0, number
          jal my_strlen
          move a2, v0                             # Cargo len(number)
          lw a0, 16($fp)                          # Cargo fd
          la a1, number                           # Cargo number
          jal my_fprintf                          # Imprimo "y_res"
          lw a0, 16($fp)                          # Cargo fd
          la a1, new_line                         # Cargo "\n"
          li a2, 1                                # Cargo len("\n")
          jal my_fprintf                          # Imprimo "\n"
          lw t0, 64($fp)
          lw a0, 40(t0)                           # Cargo shades
          subu a0, a0, 1                          # shades-1
          jal int_to_str                          # shades-1 -> str
          la a0, number
          jal my_strlen
          move a2, v0                             # Cargo len(shades-1)
          lw a0, 16($fp)                          # Cargo fd
          la a1, number                           # Cargo number
          jal my_fprintf                          # Imprimo "shades-1"
          lw a0, 16($fp)                          # Cargo fd
          la a1, new_line                         # Cargo "\n"
          li a2, 1                                # Cargo len("\n")
          jal my_fprintf                          # Imprimo "\n"
```

```
          move t0, zero
          sw t0, 20($fp)                        # Salvo y
          lw t1, 64($fp)
          lwc1 $f20, 4(t1)                      # Obtengo UL_im
          swc1 $f20, 36($fp)                    # Salvo ci = UL_im
          b test_for1
for1:
          move t0, zero
          sw t0, 24($fp)                        # Salvo x
          lw t1, 64($fp)
          lwc1 $f20, 0(t1)                      # Obtengo UL_re
          swc1 $f20, 32($fp)                    # Salvo cr = UL_im
          b test_for2
for2:
          lwc1 $f20, 32($fp)                    # Obtengo cr
          swc1 $f20, 40($fp)                    # Salvo zr = cr
          lwc1 $f20, 36($fp)                    # Obtengo ci
          swc1 $f20, 44($fp)                    # Salvo zi = ci

          move t0, zero
          sw t0, 28($fp)                        # Salvo c
          b test_for3
for3:
          lwc1 $f20, 40($fp)                    # Obtengo zr
          lwc1 $f22, 44($fp)                    # Obtengo zi
          mul.s $f24, $f20, $f20                # zr * zr
          mul.s $f26, $f22, $f22                # zi * zi
          add.s $f28, $f24, $f26                # zr * zr + zi * zi
          li.s $f30, 4
          c.le.s $f30, $f28                     # 4 <= absz
          bc1t print_pixel                      # break

          lw t1, 64($fp)
          lwc1 $f30, 24(t1)                     # Obtengo s_re
          add.s $f30, $f30, $f24                # s_re + zr * zr
          sub.s $f30, $f30, $f26                # s_re + zr * zr - zi
               * zi
          lw t1, 64($fp)
          lwc1 $f24, 28(t1)                     # Obtengo s_im
          mul.s $f26, $f20, $f22                # zr * zi
          li.s $f20, 2
          mul.s $f26, $f26, $f20                # zr * zi * 2
          add.s $f22, $f24, $f26                # s_im + zr * zi * 2
          swc1 $f30, 40($fp)                    # zr = tr
          swc1 $f22, 44($fp)                    # zi = ti

          lw t0, 28($fp)                        # Obtengo c
          addiu t0, t0, 1                       # ++c
          sw t0, 28($fp)                        # Salvo c
test_for3:
          lw t0, 64($fp)
          lw t0, 40(t0)                         # Obtengo shades
          lw t1, 28($fp)                        # Obtengo c
          subu t2, t0, t1                       # shades - c
          bgtz t2, for3

print_pixel:
          lw a0, 28($fp)                        # Obtengo c
          jal int_to_str                        # c -> str
          la a0, number
          jal my_strlen
          move a2, v0                           # Cargo len(number)
          lw a0, 16($fp)                        # Cargo fd
          la a1, number                         # Cargo number
          jal my_fprintf                        # Imprimo "c"
          lw a0, 16($fp)                        # Cargo fd
          la a1, new_line                       # Cargo "\n"
          li a2, 1                              # Cargo len("\n")
          jal my_fprintf                        # Imprimo "\n"

          lw t0, 24($fp)                        # Obtengo x
          addiu t0, t0, 1                       # ++x
          sw t0, 24($fp)                        # Salvo x
```

```
          lwc1 $f20, 32($fp)                    # Obtengo cr
          lw t2, 64($fp)
          lwc1 $f22, 16(t2)                     # Obtengo d_re
          add.s $f20, $f20, $f22                # cr += d_re
          swc1 $f20, 32($fp)                    # Salvo cr
test_for2:
          lw t0, 64($fp)
          lw t0, 32(t0)                         # Obtengo x_res
          lw t1, 24($fp)                        # Obtengo x
          subu t2, t0, t1                       # x_res - x
          bgtz t2, for2

          lw t0, 20($fp)                        # Obtengo y
          addiu t0, t0, 1                       # ++y
          sw t0, 20($fp)                        # Salvo y
          lwc1 $f20, 36($fp)                    # Obtengo ci
          lw t2, 64($fp)
          lwc1 $f22, 20(t2)                     # Obtengo d_im
          sub.s $f20, $f20, $f22                # ci -= d_im
          swc1 $f20, 36($fp)                    # Salvo ci
test_for1:
          lw t0, 64($fp)
          lw t0, 36(t0)                         # Obtengo y_res
          lw t1, 20($fp)                        # Obtengo y
          subu t2, t0, t1                       # y_res - y
          bgtz t2, for1

return:
          lw ra, 56(sp)
          lw $fp, 52(sp)
          lw gp, 48(sp)
          addu sp, sp, 64
          jr ra
          .end mips32_plot
          .size mips32_plot, .-mips32_plot

          .ent int_to_str
int_to_str:
          .frame $fp, 40, ra
          .set noreorder
          .cpload t9
          .set reorder
          subu sp, sp, 40
          .cprestore 24
          sw $fp, 28(sp)
          sw ra, 32(sp)
          move $fp, sp
          sw a0, 40($fp)                        # Fin de creacion de
              SF
          la a1, number                         # Cargo array
          la a2, index                          # Cargo indice
          jal dig_to_char                       # Convierto digitos a
               char
          la a0, number                         # Cargo array
          la a1, index                          # Cargo indice
          jal put_end                           # \0 al final
          lw ra, 32(sp)
          lw $fp, 28(sp)
          lw gp, 24(sp)
          addu sp, sp, 40
          jr ra
          .end int_to_str
          .size int_to_str, .-int_to_str

          .ent dig_to_char
dig_to_char:
          .frame $fp, 40, ra
          .set noreorder
          .cpload t9
          .set reorder
          subu sp, sp, 40
          .cprestore 24
          sw $fp, 28(sp)
          sw ra, 32(sp)
```

```
          move $fp, sp
          sw a0, 40($fp)
          sw a1, 44($fp)
          sw a2, 48($fp)                                  # Fin de creacion de
              SF
          beqz a0, return_dig
          remu t0, a0, 10                                 # r = n % 10
          sw t0, 16($fp)
          divu a0, a0, 10
          jal dig_to_char
          lw t0, 16($fp)
          lb t1, 0(a2)
          addu t2, a1, t1
          addiu t0, t0, 48                                # ascii del numero
          sb t0, 0(t2)
          addiu t1, t1, 1
          sb t1, 0(a2)
return_dig:
          lw ra, 32(sp)
          lw $fp, 28(sp)
          lw gp, 24(sp)
          addu sp, sp, 40
          jr ra
          .end dig_to_char
          .size dig_to_char, .-dig_to_char

          .ent put_end
put_end:
          .frame $fp, 8, ra
          .set noreorder
          .cpload t9
          .set reorder
          subu sp, sp, 8
          .cprestore 0
          sw $fp, 4(sp)
          move $fp, sp
          sw a0, 8($fp)
          sw a1, 12($fp)                                   # Fin de creacion de
               SF
          lb t0, 0(a1)
          addu t0, a0, t0
          sb zero, 0(t0)
          sb zero, 0(a1)                                   # Reinicio indice
          lw $fp, 4(sp)
          lw gp, 0(sp)
          addu sp, sp, 8
          jr ra
          .end put_end
          .size put_end, .-put_end

          .ent my_strlen
my_strlen:
          .frame $fp, 8, ra
          .set noreorder
          .cpload t9
          .set reorder
          subu sp, sp, 8
          .cprestore 0
          sw $fp, 4(sp)
          move $fp, sp
          sw a0, 8($fp)                                   # Fin de creacion de
              SF
          move t0, zero                                   # i = 0
          b test_end
increment:
          addiu t0, t0, 1                                 # i++
test_end:
          addu t1, t0, a0
          lb t1, 0(t1)
          bnez t1, increment                              # if(s[i] != 0) ->
              increment
          move v0, t0
          lw $fp, 4(sp)
          lw gp, 0(sp)
```

```
          addu sp, sp, 8
          jr ra
          .end my_strlen
          .size my_strlen, .-my_strlen

          .ent my_fprintf
my_fprintf:
          .frame $fp, 40, ra
          .set noreorder
          .cpload t9
          .set reorder
          subu sp, sp, 40
          .cprestore 24
          sw $fp, 28(sp)
          sw ra, 32(sp)
          move $fp, sp
          sw a0, 40($fp)
          sw a1, 44($fp)
          sw a2, 48($fp)                                  # Fin de creacion de
              SF
          move t0, zero                                   # n = 0
          move t1, zero                                   # total = 0
          sw t0, 16($fp)                                  # n -> SF
          sw t1, 20($fp)                                  # total -> SF
          b test_write
add_total:
          lw t1, 20($fp)
          lw t0, 16($fp)
          addu t1, t1, t0
          sw t0, 16($fp)
          sw t1, 20($fp)
test_write:
          lw a0, 40($fp)                                  # Cargo fd
          lw t3, 44($fp)
          lw t2, 48($fp)
          lw t1, 20($fp)
          lw t0, 16($fp)
          addu a1, t3, t1                                 # Cargo dir + total
          subu a2, t2, t1                                 # Cargo len - total
          li v0, SYS_write
          syscall
          move t0, v0
          sw t0, 16($fp)
          bgtz t0, add_total
          lw v0, 20($fp)                                  # Devuelvo el total
          lw ra, 32(sp)
          lw $fp, 28(sp)
          lw gp, 24(sp)
          addu sp, sp, 40
          jr ra
          .end my_fprintf
          .size my_fprintf, .-my_fprintf

          .rdata
          .align 2
msgs:     .word head_pgm, new_line
          .align 0
head_pgm: .asciiz "P2\n"
new_line: .asciiz "\n"

          .data
number:   .space 11
index:    .byte 0
```

5. Bibliografía

1. GXemul.
   http://gavare.se/gxemul/.

2. The NetBSD project.

http://www.netbsd.org/.

3. Conjunto de Julia
   http://es.wikipedia.org/wiki/Conjunto_de_Julia (Wikipedia).

4. PGM format specification.
   http://netpbm.sourceforge.net/doc/pgm.html.

5. Generador de fractales.
   http://usefuljs.net/fractals/

6. GIMP.
   https://www.gimp.org/

7. System V Application Binary Interface.
   http://math-atlas.sourceforge.net/devel/assembly/mipsabi32.pdf